**PARTE I – ORIENTAÇÕES**

**Duração da Prova**: **03 hs contínuas (ao iniciar a prova o aluno deve finalizar a mesma sem interrupções).**

**Tipo de Prova**: Individual, com consulta permitida a sites e livros, não podendo ser compartilhado resultados.

**Forma de submissão**:

1. Toda a submissão será no GITHUB
2. Cada questão deverá ser um projeto completo diferente no GITHUB, inclusive as questões teóricas.

**PARTE II – TEORIA**

**ORIENTAÇÃO: RESPONDA AS QUESTÕES E SUBMETA O ARQUIVO EM SEU REPOSITÓRIO GITHUB.**

I - Em POO (Programação Orientada a Objetos), dizer que a classe A estende a classe B é o mesmo que dizer que:

1. a classe B é subclasse de A;
2. a classe A é superclasse de B; X
3. a classe A é derivada de B;
4. a classe B é derivada de A;
5. as classes A e B são irmãs.

II - A assinatura dos métodos de uma classe é composta por nome do método, tipo de retorno, lista de argumentos e visibilidade.

[ x ] Certo

[ ] Errado

III - Em programação orientada a objetos, a possibilidade de haver funções de mesmo nome, com funcionalidades similares em classes sem nenhuma relação entre elas, denomina-se

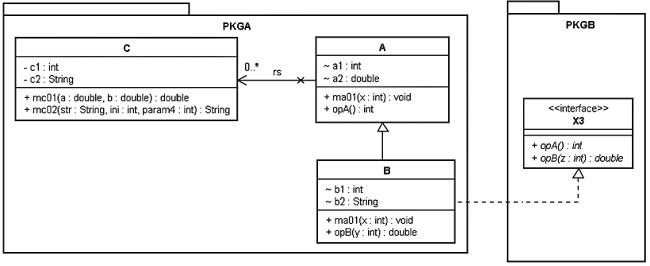
1. encapsulamento. X
2. objeto.
3. classe.
4. polimorfismo.
5. relacionamento hierárquico.

IV - Sobre Programação Orientada a Objetos, analise:

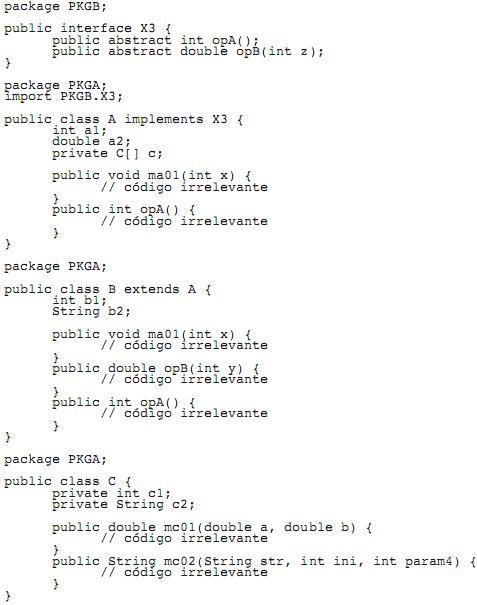
1. A encapsulação garante que apenas as interfaces necessárias para interação com o objeto estejam visíveis, e atributos internos não sejam acessíveis.
2. O polimorfismo garante que objetos possam herdar métodos e atributos de uma superclasse para a geração de uma nova classe.
3. A herança possibilita que distintas operações na mesma classe tenham o mesmo nome, desde que alterada a assinatura.

Está correto o que se afirma em

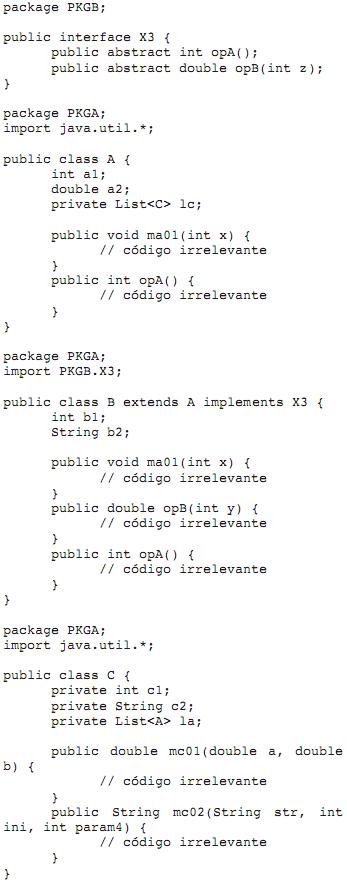
* 1. III, apenas.
  2. II, apenas.
  3. I, apenas. X
  4. II e III, apenas.
  5. I, II e III.

**V - Seja o seguinte diagrama de classes UML:**  
  
**Qual conjunto de classes e interfaces Java é compatível com os elementos de modelagem existentes nesse diagrama?**

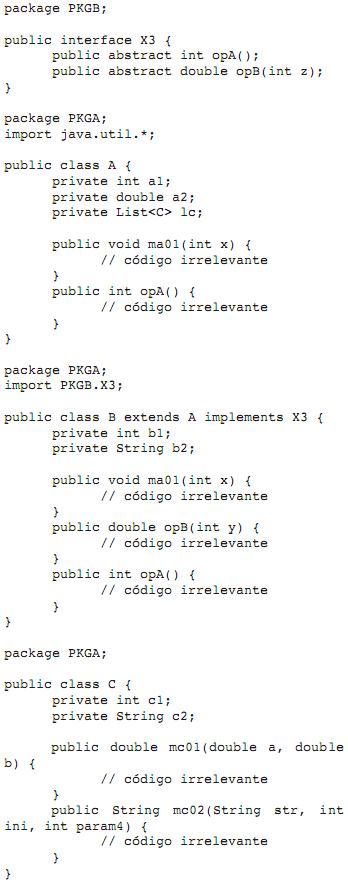
**(a)**



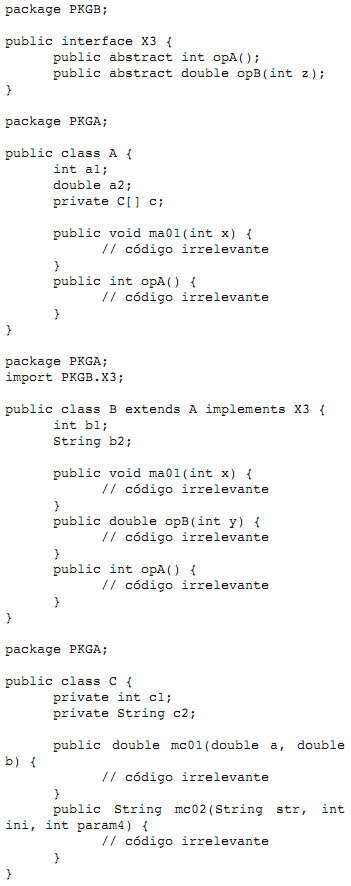
**b) X**



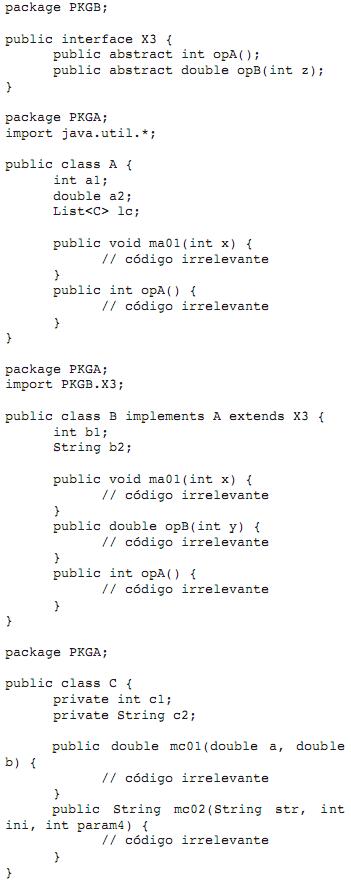
**c)**



**d)**



**e)**



VI - Na programação orientada a objetos, há dois mecanismos possíveis para criar relacionamentos entres classes: a composição e a herança. Em um programa orientado a objetos, suponha que entre duas classes X e Y exista um relacionamento de herança no qual X é subclasse de Y. Suponha também que Y seja uma subclasse de uma classe abstrata denominada Z. Por fim, suponha que existam apenas métodos públicos ou privados, tanto em Y quanto em Z.

Para alterar esse programa de tal forma que ele permaneça o mesmo, do ponto de vista funcional para os clientes da interface pública da classe X, e que o relacionamento de herança entre X e Y seja substituído pelo de composição, um passo necessário no caso geral é definir um atributo do tipo

1. X na classe Y
2. X na classe Z
3. Y na classe X
4. Y na classe Z
5. Z na classe X

VII - Analise as seguintes definições:

1. Objeto que existe depois que o processo ou o thread que o criou deixa de existir.
2. Mecanismo pelo qual elementos mais específicos incorporam a estrutura e o comportamento de elementos mais gerais.
3. Lista de valores nomeados, utilizada como faixa de um determinado tipo de atributo.

Correspondem, respectivamente, às definições I, II e III:

1. objeto transiente, método e cardinalidade.
2. objeto ativo, polimorfismo e multiplicidade.
3. objeto transiente, polimorfismo e multiplicidade.
4. objeto persistente, herança e enumeração.
5. objeto ativo, mensagem e enumeração.

**PARTE III – PRÁTICA**

**QUESTÃO 1 – Dado o código contido no repositório abaixo, analise o mesmo e identifique possíveis partes de códigos inseguros. De posse disso, através de técnicas de programação segura, ajuste o mesmo de forma a eliminar todos os mau-cheiros, bem como vulnerabilidades encontradas no mesmo. (2.0 PONTOS) ATENTE PARA AS OBSERVAÇÕES:**

OBS1 - Repositório: <https://github.com/kabartsjc/p2exe02.git>

OBS2 – Deve ser analisado o código para quaisquer vulnerabilidades possíveis.

OBS3 – Devem ser criados testes automatizados que demonstre que o código não pode ser maliciosamente manipulado.

OBS4: O Projeto deve ser um projeto Maven, onde todas as bibliotecas necessárias devem ser configuradas dentro do POM (a correção vai ser binária, caso o projeto não compile ao se rodar o POM).

**QUESTÃO 2 - Considere um sistema de busca de pacotes de viagem online, onde um Pacote de Viagem é composto por:**

1. **- Nome do pacote;**
2. **- Número de Dias do Pacote;**
3. **- Região a ser visitada;**
4. **- Cia Aérea;**
5. **- Qtde de Horas do Voo;**
6. **- Preço.**

**Nesse Sistema o usuário ao clicar no botão buscar, deve receber como resposta um conjunto de pacotes ordenados por um dos seguintes critérios:**

1. **Menor Valor do pacote.**

TAREFA 1 – DESENVOLVA O SUPRACITADO SISTEMA (NÃO É NECESSÁRIA INTERFACE GRÁFICA).

TAREFA 2 – DESENVOLVA CASOS DE TESTES AUTOMATIZADO (XUNIT) QUE PERMITA VALIDAR OS CENÁRIOS PROPOSTOS PELO USUÁRIO.

**QUESTÃO 3 – Construa uma calculadora que seja capaz de realizar as quatro operações básicas (+ - \* \). Construa cenários de testes automatizados (xunits) que permitam validar a mesma.**

**QUESTÃO 4 - Verifique e explique se o código Java abaixo segue a Lei de Demeter (LoD - Law of Demeter) ou não. Se não segue, refatore-o adequadamente. Use os números das linhas para comentar qualquer refatoração. Explique especificamente porque a linha 3, a que está comentada com um (\*), segue ou não a LoD. Se não a segue, também explique porque a refatoração realizada em (a) a segue.**

**public** **class** LawOfDemeterInJava {

private CheeseToping \_cheeseToping;

...

**public** **void** examples(Pizza pizza, String topingDescription)

{

Foo foo = **new** Foo();

1 doSomething();

**2 int** price = pizza.getPrice();

3 String topingName = topingDescription.trim().toLowerCase().substring(0,10).trim(); //(\*)

4 \_cheeseTopping = **new** CheeseTopping(topingName);

**5 float** weight = \_cheeseTopping.getWeightUsed();

6 foo.doBar();

…

}

**QUESTÃO 5 - Talvez você pense em acrescentar um método para a interface do Messenger (ver código abaixo). Mas isso faz com que você recompilar todos os usuários dessa interface e quase todo o sistema precisa ser reimplantado.**

1. **Dentro dos princípios do SOLID, qual(is) foi(ram) violado(s)? Explique.**

public interface Messenger {

askForCard();

tellInvalidCard();

askForPin();

tellInvalidPin();

tellCardWasSiezed();

askForAccount();

tellNotEnoughMoneyInAccount();

tellAmountDeposited();

tellBalance();

}

1. **Ainda considerando princípios SOLID, como resolver o problema? Use o(s) princípio(s) indicados e modifique o código acima apresentado.**